Dokumentation COM-Schnittstelle Warenwirtschaft Version 13.0

Warenwirtschaft

Optimieren Sie Ihre Geschäftsabläufe und

lassen Sie die Software für sich arbeiten.

SelectLine®
Kaufmännische Software

Inhaltsverzeichnis

1 Änderungen

1.1 Wawi 13.2.0

Neu:

- Funktion **IApp.BelegManuellErledigen(BelegTyp: WideString; BelegNummer: WideString);**
  Mit der Com-Funktion "BelegManuellErledigen" lassen sich offene Belege aus der Belegkette und Werkaufträge manuell erledigen. Als Parameter sind der Belegtyp und die Belegnummer zu übergeben.

1.2 Wawi 13.0.1

Neu:

- Funktion **IOlePos.SpeichernMitLagerinformation(Typ: Integer; const Lagerinformation: WideString)** zum Speichern von Belegpositionen per COM. Die Funktion hat zwei Parameter. Als erster Parameter ist eine 0 einzugeben. Dieser Parameter wird zurzeit nicht ausgewertet. Als zweiter Parameter ist ein String (mit Zeilenumbrüchen) anzugeben, der die Lagerinformationen zur Belegposition enthält. Pro Zeile sind dort - mit Semikolon getrennt - die Menge, das Lager, die Serien-/Chargennummer, das Verfallsdatum und die Preismenge einzutragen. Enthält das Lager oder die Seriennummer ein Semikolon, kann der Eintrag mit " umschlossen werden.

1.3 Wawi 13.0.0

Neu:

- Funktion **IApp.EMailKontaktAnlegen**
  Rückgabewert Integer -> String

1.4 Wawi 12.0.2.9

Neu:

- Funktion **IWerkauftrag.Auslagern** zum Auslagern von Stücklistenunterartikeln für die Produktion.

Änderungen:

- Die Funktion **IWerkauftrag.SerieChargeListeEinlagern** setzte in den Versionen 12.0 bis 12.0.2.8 die Fertigungsmengen für "In Produktion" in der Tabelle LAGERFERTIGUNG des Werkauftrages nicht zurück. Der Fehler wurde korrigiert.
- Die Funktionen **XFakt.LiesOptionen** und **XFakt.SetzeOptionen** wurden an die neuen Mandanteneinstellungen der Version 12 angepaßt.
- Die Auslagerung von S/C-Artikeln mit den COM-Funktionen **OleBeleg.BucheSeriennummern** und **OlePos.SetArtikel** (mit Angabe einer S/C-Nr.) funktionierte in den Versionen 12.0 bis 12.0.2.8 nicht. Der Fehler wurde korrigiert.

1.5 Wawi 12.0.2.8

Änderungen:

- Der Parameter Mengenanpassung der Funktion **XFakt.WandelBelegEx2** erlaubt die Angabe von Verfallsdatum und Preismenge pro Mengenanpassungszeile.

1.6 Wawi 12.0.2

Änderungen:

- Die Funktion **XFakt.Umlagern** wurde an die neuen Lagerungsstrukturen angepaßt. Bei jedem Aufruf wird zur Zeit eine neue manuelle Lagerung erstellt.

1.7 Wawi 12.0

Änderungen:

- Zur Unterstützung der neuen GUID-Beleg(positions)kennungen wurde der Datentyp Integer in WideString bei folgenden COM-Funktionen geändert:**IOleBeleg.EditPos**, **IOleBeleg.DeletePos**, **IOleBeleg.CreateUnterPos**, **IOlePos.Save**, **IOlePos.GetKennung**
- Die Unterstützung für StampIt wurde entfernt. Die Konstante "lizenzStampit" ($0000000C) wird von **XFakt.HatLizenz** nicht mehr unterstützt.
- Die Versionsnummer des COM-Interfaces wurde von 11.5 auf 12.0 erhöht.
- *Sonstige Änderungen betreffen nur interne Funktionalitäten zum Zugriff auf das SelectLine Outlook-Add-In.*

1.8 Wawi 11.5.1

Alle Änderungen betreffen nur interne Funktionalitäten für die SelectLine Rewe-Wawi-COM-Kopplung.

1.9 Wawi 11.5

Neu:

- Property **IOlePos.Bestellnummer** zum Zugriff auf die Bestellnummer der Belegposition.

Änderungen:

- Die Versionsnummer des COM-Interfaces wurde von 11.3 auf 11.5 erhöht.

1.10 Wawi 11.3

Neu:

- Funktionen **XFakt.LeseArtikelInaktivStatus** und **XFakt.SetzeArtikelInaktivStatus** zum Setzen des Inaktiv-Flags am Artikel.

Änderungen:

- Die Versionsnummer des COM-Interfaces wurde von 11.2 auf 11.3 erhöht.
- *Sonstige Änderungen betreffen nur interne Funktionalitäten zum Zugriff auf das SelectLine Outlook-Add-In.*

1.11 Wawi 11.2

Änderungen:

- Die Versionsnummer des COM-Interfaces wurde von 11.1 auf 11.2 erhöht.
- *Sonstige Änderungen betreffen nur interne Funktionalitäten zum Zugriff auf das SelectLine Outlook-Add-In.*

1.12 Wawi 11.1

Änderungen:

- Die Versionsnummer des COM-Interfaces wurde von 11.0 auf 11.1 erhöht.
- *Sonstige Änderungen betreffen nur interne Funktionalitäten zum Zugriff auf das SelectLine Outlook-Add-In.*

1.13 Wawi 11.0.1

Neu:

- Funktion **XFakt.BilderExportEx** zum Exportieren von Bildern aus der Tabelle BILD. Das Delphi-Demo wurde entsprechend erweitert.

Änderungen:

- Die Funktionen **OleBeleg.AddFusstextLine** und **OleBeleg.AddKopftextLine** wurden korrigiert. In der Wawi V. 11.0 war die Funktionalität im Zuge der Unicode-Umstellung defekt.
- Mit der Funktion **XFakt.Umlagern** können jetzt auch Serien- und Chargennummern umgelagert werden, wenn diese anstelle der Artikelnummer übergeben werden.

1.14 Wawi 11.0

Änderungen:

- Mit der Funktion **XFakt.LiesOptionen** sind Abfragen der einzelnen Teile der Versionsnummer möglich. Die Optionen sind unter **XFakt.SetzeOptionen** beschrieben.
- Das Interface IOleAdress für die Kommunikation mit dem nicht mehr unterstützten Cobra-Provider wurde entfernt.
  Die Funktion **XFakt.CreateOleAdress** gibt ein IUnknown-Interface zurück.
- Das Verhalten der COM-Komponente **OleTable** wurde überarbeitet und hinsichtlich SQL optimiert. Dadurch hat sich auch das Verhalten von **OleTable.RecordCount()** minimal geändert (s. Abschnitt „Änderungen an OleTable ab Wawi V. 11.0“).
- Die Versionsnummer des COM-Interfaces wurde von 10.0 (Wawi V. 10.x.x) auf 11.0 (Wawi V. 11.0) erhöht.

Ergänzung der Dokumentation:

- Der Abschnitt Tipps & Tricks ist neu und enthält u. a. das Thema "Über COM aufgerufene Dialoge erhalten keinen Fensterfokus und öffnen sich im Hintergrund". Das Delphi-COM-Demo wurde erweitert.

1.15 Wawi 10.1.7

Änderungen:

- Erweiterung der Funktion **XFakt.SetzeOptionen** um die Möglichkeit, beim Druck von Umlagerungsbelegen über **XFakt.DruckeBeleg** den Sortierreihenfolge-Dialog zu umgehen.

1.16 Wawi 10.1.6

Änderungen:

- Die Funktion XFakt.DruckeBeleg verwendete für Umlagerungsbelege den Standardbelegdruck, so dass Quell- und Ziellager nicht ausgedruckt wurden. Der Fehler wurde korrigiert, so dass Umlagerungsbelege jetzt korrekt gedruckt werden können.
- Der Druck von Umlagerungsbelegen mit der Funktion **XFakt.DruckeBelegMitVorlage** steht nicht mehr zur Verfügung, da für Umlagerungsbelege ein internes Sonderhandling notwendig ist. Zum Ausdruck dieser Belege muß die Funktion XFakt.DruckeBeleg verwendet werden.
- Die Funktion **XFakt.Umlagern** legte bisher einen Belegkopf für einen Umlagerungsbeleg an, wobei die Belegnummern des Belegkopfes und der Umlagerungsposition nicht übereinstimmten. Dadurch wurden Umlagerungsbelege ohne Positionen erstellt. Auf die Ansicht Historie in den Artikelstammdaten sowie auf die Funktionalität hatte dies keinen Einfluß. Das Verhalten der Funktion wurde dahingehend angepaßt, dass sich diese Funktion jetzt wie eine manuelle Umlagerung (Artikelstammdaten, Seite Lager) verhält. Es wird kein Umlagerungsbeleg mehr erstellt (Tabelle BELEG), sondern nur noch (manuelle) Umlagerungspositionen (Tabelle BELEGP, Belegtyp "U").

1.17 Wawi 10.1.4

Neu:

- Für den Druck von Belegen steht die Funktion **XFakt.DruckeBelegMitVorlage** zur Verfügung, der die Druckvorlage sowie weitere Optionen übergeben werden können

Änderungen:

- Die Verwendung der beiden Funktionen **IOlePos.SetEingabeMenge** und **IOlePos.SetArtikel** zum Einlagern von Chargenartikeln öffnete - unabhängig von Mandanteneinstellungen - immer den Einlagerungsdialog mit vorbelegter Menge 1. Der Fehler wurde behoben.

1.18 Wawi 10.1.3

Neu:

- Funktion **XFakt.BilderExport** zum Exportieren von Bildern aus der Tabelle BILD.

1.19 Wawi 10.1.2

Neu:

- Funktion **XFakt.LiesOptionen** zum Auslesen von temporär gesetzten Mandanteneinstellungen.

Änderungen:

- Erweiterung der Funktion **XFakt.SetzeOptionen** um temporäre Mandanteneinstellungen für die Ein- und Auslagerungsdialoge.
- Mit der Funktion **XFakt.Umlagern** konnte bisher nicht aus Wareneingangslägern umgelagert werden. Das ist ab sofort möglich.

1.20 Wawi 10.1.1

Neu:

- Property **IOleBeleg.Lager** zum Zugriff auf das Beleglager.

1.21 Wawi 10.1

Neu:

- Funktion **IWerkauftrag.AktionMengeDatum** zur Ausführung von Werkauftragsaktionen mit Mengen- und Datumsangabe.
- Property **IWerkauftrag.Artikelnummer** zur Abfrage des Werkauftragartikels.
- Property **IWerkauftrag.Menge** zur Abfrage der Werkauftragsmenge.

1.22 Wawi 10.0.4

Neu:

- Mit der Funktion **XFakt.HatLizenz** können die freigeschalteten Lizenzen der registrierten COM-Wawi abgefragt werden.

1.23 Wawi 10.0

Neu:

- Property **XFakt.MandantLand** zur Abfrage des Mandantenlandes.
- Property **XFakt.SQLServer** zur Abfrage des verbundenen SQL Servers.
- Prozedur **XFakt.StartMakroEx** sowie das Interface **IMakroParamList** zum Ausführen von Makros mit Parameterübergabe.
- Funktion **IWerkauftrag.SerieChargeListeEinlagern** zum Einlagern von Serien-/Chargennummern im Werkauftrag
- Property **IOleBeleg.AutoAddZubehoer** zur Steuerung des Zufügens von Zubehörartikeln.
- Property **IOleBeleg.AutoAddZuschlag** zur Steuerung des Zufügens von Zuschlagartikeln.

Änderungen:

- Der Parameter Level der Funktion **IOlePos.Save** wird ignoriert. Das Erzeugen von Unterpositionen ist nur noch über den Parameter AHauptkennung der Funktion IOleBeleg.CreateUnterPos möglich.
- Die Versionsnummer des COM-Interfaces wurde von 1.5 (Wawi V. 9.0.x) auf 10.0 (Wawi V. 10.0) erhöht.

1.24 Wawi 9.0.6

Änderungen:

- Bei der Belegübergabe mit der Funktion **XFakt.WandelBelegEx** wurde der Beleg intern immer als Sammelbeleg behandelt. Deshalb wirkten unter Umständen Mandantenoptionen, z. B. "Mindestwert für Umsatzsammelbeleg" (Seite Belege | Übernahme / Statistik). Wenn der Umsatz unter dem hier angegebenen Wert lag, scheiterte die Belegübergabe. Der Fehler wurde behoben.

1.25 Wawi 9.0.5

Neu:

- Property **IWerkauftrag.BelegNummer** zum Zugriff auf die Belegnummer des Werkauftrags.

1.26 Wawi 9.0.4

Änderungen:

- nur BDE-Version: **XFakt.Login** stellt im Modus OLELOGIN=0 automatisch den mit dem Nutzerkürzel zuletzt benutzten Mandanten ein.
- **XFakt.Umlagern** unterstützt jetzt auch Läger mit Lagerplätzen.
- Der Funktion **IOlePos.SetArtikel** kann statt einer Artikelnummer oder EAN jetzt auch eine gültige Serien-/Chargennummer übergeben werden.
- Die Funktion **ILager.InventurIstWert** unterstützt jetzt auch Läger mit Lagerplätzen.

1.27 Wawi 9.0.3

Neu:

- **IOleBeleg.SetStandort** setzt den Standort für den Beleg und schränkt damit die verfügbaren Läger auf diesen Standort ein.

Änderungen:

- **IOlePos.SetLager** wurde dahingehend erweitert, dass jetzt zusätzlich die Angabe des Lagerplatzes möglich ist.
- **XFakt.WandelBelegEx2**: Als zusätzlicher Parameter in der Mengenanpassungsliste ist die Angabe von Lager und Lagerplatz möglich.

1.28 Wawi 9.0

Neu:

- neuer Kommandozeilenparameter **/REGINFO** zur Abfrage der COM-Server-Registrierung
- Prozedur **XFakt.StartMakro** zum Ausführen von Makros.
- Properties **IOleBeleg.BelegRundung**, **IOleBeleg.PositionsRundung** zum Lesen/Schreiben der Belegrundungsmöglichkeiten.

Änderungen:

- Das Feld **Lager** der Artikeltabelle **ART** wird nicht mehr benutzt.Bis zur Version 8.2.x diente dieses Feld zur Speicherung des Standardlagers eines Artikels. Ab der Version 9.0 sind pro Artikel mehrere Standardläger möglich, die in der neuen Tabelle ARTORTLAGER verwaltet werden. Das Feld Lager in ART wird bei neu angelegten Artikeln nicht mehr gefüllt und ist deshalb NULL.
- Die Funktion **XFakt.EditBeleg** generiert beim Zugriff auf nicht existierende Belege eine Exception.Bis zur Version 8.2.x stürzte die Warenwirtschaft dabei ab.
- **IOlePos.Save** kann auch mit den neu eingeführten Zuschlagartikeln umgehen.Nach dem Speichern einer Belegposition eines Artikels, für den Zuschlagartikel angelegt sind, werden diese automatisch eingefügt. Danach kann in genauso begrenztem Maße auf diese Zuschlagartikel zugegriffen werden, wie beim händischen Erstellen eines Beleges.
- **IOlePos.SetLager** überprüft, ob das Lager zum eventuell schon gesetzten Standort paßt.Im Fehlerfall wird eine Exception generiert, sonst wird automatisch der Standardlagerplatz des Lagers (der erste Lagerplatz) eingestellt. Lagerplätze werden zur Zeit noch nicht über das COM-Interface unterstützt, können also nicht gezielt ausgewählt werden. Bei der Belegerstellung (z. B. Lagerung - Wareneingang) laufen intern dieselben Automatismen ab, wie bei der interaktiven Benutzung der Wawi (z. B. Einstellung Lager Einlagerungsautomatik "bis Maximalmenge füllen", Lagerplatzreservierung für bestimmte Artikel, Maximalmengenbegrenzung usw.).

1.29 Wawi 8.bis Wawi 8.2.11

Änderungen:

- IOleBeleg.PutField: Datumswerte können als WideString übergeben werden.Bisher mußten Datumswerte „kryptisch“ als Zahl im internen Format an diese Funktion übergeben werden. Jetzt ist die Übergabe „richtiger“ Datumswerte als WideString möglich (z. B. „31.12.2006“)
- **XFakt.WandelBelegEx2**: In der übergebenen Stringliste können statt Seriennummern auch Chargennummern verwendet werden.

1.30 Wawi 8.2.3

erster Stand der COM-Schnittstellenbeschreibung

2 Einführung

Die COM-Schnittstelle bietet bis jetzt folgenden Funktionsumfang:

- Mit dem Tabellenobjekt können Datensätze gelesen, editiert sowie neue Datensätze zugefügt werden.
- Das Belegobjekt beherrscht die Erstellung von Belegen.
- Das Positionsobjekt erlaubt das Anlegen von Belegpositionen.
- Weiterhin besteht die Möglichkeit, interaktiv einen Mandanten sowie beliebige Stammdaten aus Listen auszuwählen und zu bearbeiten, Belege zu bearbeiten, zu drucken und zu übergeben. Eine detaillierte Beschreibung aller Funktionen finden Sie in der Beschreibung des Fakt-Objektes.

Installation des COM-Servers:

Die Warenwirtschaft registriert sich durch einen einmaligen Aufruf der Exe-Datei mit dem Parameter `/regserver`.

Deinstallation des COM-Servers:

Zum Deinstallieren des Server-Objekts ist das Programm mit dem Parameter `/unregserver` aufzurufen.

Registrierungsinfo:

Mit dem Parameter `/reginfo` erscheint eine Dialogbox mit den aktuellen Registrierungseinträgen in der Windows-Registrierung.

Ab Version 12.x können Sie die COM Registrierung auch direkt im Programm vornehmen: *„Applikationsmenü - Programmeinstellungen - COM Server"*

3 Fakt-Objekt - XFakt

Dieses Objekt beinhaltet Funktionen für das Anlegen von Tabellen- und Belegobjekten sowie die Daten- und Mandantenauswahl. Es wird innerhalb des Automatisierungscontrollers mit dem Namen "XFakt.App" initialisiert.

Beispiel für Visual Basic:

```
Set Fakt = CreateObject("XFakt.App")
While Not Fakt.ApplicationReady
Wend
Fakt.Login("Nutzerkürzel", "Passwort")
```

Die folgenden Funktionen beziehen sich dann beim Aufruf auf dieses Objekt.

Beispiel für Delphi:

```
Table := Fakt.CreateTable('KU');
```

Funktion	Beschreibung
`ApplicationReady: OleVariant`	Überprüft, ob das Programm vollständig gestartet wurde.
`CreateTable(const ABlobKey: WideString):OleTable`	Diese Funktion initialisiert ein Tabellenobjekt. Der Parameter *TableKey* enthält einen zweistelligen String aus Grossbuchstaben um eine Tabelle auszuwählen. Das Handling, ob es sich dabei um Tabellen aus dem Daten- oder Mandantenverzeichnis handelt, wird dadurch gekapselt. Die Liste der Schlüssel befindet sich im Anhang.
`CreateQuery(ADataBaseName: WideString): OleQuery`	Diese Funktion initialisiert ein Queryobjekt. Der Parameter *ADataBaseName* enthält optional den logischen Verzeichnisnamen • XDATENDB für das Datenverzeichnis • XMANDANTDB oder leer für das aktuelle Mandantenverzeichnis

`Login(Benutzerkürzel, Passwort: WideString): WordBool`	Der Aufruf dieser Funktion ist erforderlich, wenn Benutzer angelegt sind und in der DATEN.INI (nur BDE-Version) folgender Eintrag steht:
	`[SYSTEM]`
	`OLELOGIN=0`
	Mit `OLELOGIN=1` oder Löschen des Eintrags wird das Passwort bei Programmstart über den normalen Passwort-Eingabedialog abgefragt.

`Mandant: String`	Liefert die Nummer des aktuell ausgewählten Mandanten.

`GetMandant(Mandant: WideString; Auswahl: Integer): OleVariant`	Stellt den aktuellen Mandanten ein. **Parameter:** • *Auswahl* 0: Der Mandant wird automatisch gewechselt. 1: Der Auswahldialog wird angezeigt. • *Mandant* gibt die Mandantennummer an • *Ergebnis* Die Nummer des aktiven Mandanten
`GetMandField(Feldname: string): OleVariant`	Liefert Werte von Mandantenfeldern.
`FindEinheitFromEdiEinheit(const Einheit: WideString): WideString`	Gibt die zur übergebenen Edifact-Einheit gehörende interne Einheit zurück.
`SelectData(const BlobKey: WideString; const LastKey: WideString): OleVariant`	Anzeige eines Listen-Auswahldialoges. **Parameter:** • *BlobKey* bestimmt die anzuzeigende Tabelle • *LastKey* markiert den Datensatz mit dem angegebenen Schlüssel

`EditData(const BlobKey: WideString; const Key: WideString): OleVariant`	Anzeige eines modalen Bearbeitungsdialoges.
	
	Parameter:
	• *BlobKey*
	bestimmt die anzuzeigende Tabelle
	• *Key*
	markiert den Datensatz mit dem angegebenen Schlüssel
	
	Die Funktion liefert *Key* als Rückgabewert, es sei denn der Datensatz wurde in der Bearbeitungsmaske gelöscht.
	
	Beispiele
	
	öffnet die Kundenmaske mit dem Kunden 10000:
	`EditData('KU','10000')`
	
	öffnet die Rechnungsmaske mit der Rechnung 9800123:
	`EditData('QR','9800123')`

`NeuerBeleg(const BelegTyp: WideString; const BelegNummer: WideString): OleBeleg`	Mit dieser Funktion wird ein Belegobjekt angelegt (bzw. ein Beleg erstellt). Gültige Belegtypen sind z.B.: A-Angebot, F-Auftrag, L-Lieferschein, R-Rechnung B-Bestellung, E-Eingangsrechnung. Der Parameter Belegnummer ist optional.

`NeuerBelegAuto(const BelegTyp: WideString; const BelegNummer: WideString): OleBeleg`	Mit dieser Funktion wird ein Belegobjekt angelegt (bzw. ein Beleg erstellt). Gültige Belegtypen sind z.B.: A-Angebot, F-Auftrag, L-Lieferschein, R-Rechnung B-Bestellung, E-Eingangsrechnung. Fehlt die Belegnummer wird diese vom System automatisch ohne Nachfrage vergeben.
`EditBeleg(const BelegTyp: WideString; const BelegNummer: WideString): OleBeleg`	Liefert ein Belegobjekt.
`WandelBeleg(const VonBelegTyp: WideString; const VonBelegNummer: WideString; const NachBelegTyp: WideString): WideString`	Belegübergabe. Rückgabe: neue Belegnummer
`DeleteBeleg(const BelegTyp: WideString; const BelegNummer: WideString): OleVariant`	Löscht den angegebenen Beleg und seine Positionen.
`DruckeBeleg(const BelegTyp: WideString; const BelegNummer: WideString): OleVariant`	Druckt den angegebenen Beleg. Ab Wawi 10.1.6 werden Umlagerungsbelege korrekt mit Quell- und Ziellager gedruckt.Ab Wawi Version 10.1.7 kann beim Druck von Umlagerungsbelegen durch das Setzen der COM-Umgebungs-Option "Beleg.U.Druck.Sortierung" (XFakt.SetzeOptionen) der Sortierreihenfolge-Dialog unterdrückt werden.
`function DruckeBelegMitVorlage(const Belegtyp: WideString; const Belegnummer: WideString; const Vorlage: WideString; VorlageTyp: Integer; const Optionen: WideString): Integer;`	Die Funktion druckt den angegebenen Beleg unter Verwendung der übergebenen Druckvorlage sowie der optional eingestellten Optionen. Der Rückgabewert ist 0 bei Erfolg bzw. > 0 bei einem Fehler, wobei "Erfolg" bedeutet, dass die Druckdaten ohne Fehler an das Betriebssystem weitergereicht werden konnten. Diese Funktion unterstützt außerdem den Stillen Modus. **Parameter:** • *Belegtyp* A = Angebot Kunde, R = Rechnung Belegnummer • *Belegnummer* • *Vorlage* Vorlagendateiname (ohne Punkt und Dateiendung!) oder Bezeichnung der Vorlage.Nach Möglichkeit sollte immer der Dateiname angegeben werden, da die Bezeichnung nicht eindeutig sein muß. Es ist innerhalb der Wawi möglich (aber nicht empfehlenswert!), pro Belegtyp mehrere Druckvorlagen mit derselben Bezeichnung anzulegen.Bei Angabe des Dateinamens muß die Dateiendung weggelassen werden, da diese aus dem Belegtyp automatisch ermittelt wird (z. B. ".REC" für Rechnung). Bsp.: BLATT1 = Dateiname, Rechnung = Bezeichnung

- *VorlageTyp*
  Angabe des Vorlagetyps:0 = Vorlage ist Dateiname
  1 = Vorlage ist Bezeichnung
- *Optionen*
  optionale Angabe weiterer Druckoptionen. Mit diesem Parameter können weitere Druckoptionen angegeben werden. Die Syntax entpricht der der Funktion `XFakt.SetzeOptionen:` Option und Parameter werden durch Semikolon getrennt, mehrere Optionen durch Zeilenumbruch. Folgende Optionen sind möglich:
  - *Ausgabeziel;<Integer>*
    Übersteuerung des Ausgabeziels der Druckvorlage
    -1 = Ausgabeziel aus der Druckvorlage nehmen. Ist in der Druckvorlage der Standarddrucker angegeben, so wird das Ausgabeziel aus den Mandanteneinstellungen bestimmt (gleiches Verhalten wie innerhalb der Wawi)
    0 = Drucker
    1 = Bildschirm
    2 = Datei
    3 = Zwischenablage
    4 = Email
    5 = Archiv (wenn vorhanden)
  - *DialogDruckziel;<Integer>*
    Anzeige des Ausgabezieldialoges
    -1 = Standardeinstellung (z. Zt. Anzeige des Dialoges)
    0 = Dialog nicht anzeigen
    1 = Dialog anzeigen
  - *DialogDruckereinrichtung;<Integer>*
    Anzeige des Druckereinrichtungsdialoges
    -1 = Standardeinstellung (z. Zt. Anzeige des Dialoges)
    0 = Dialog nicht anzeigen
    1 = Dialog anzeigen

Falls eine oder alle Optionen nicht angegeben werden (Leerstring übergeben), sind über COM folgende Werte voreingestellt:
Ausgabeziel;-1
DialogDruckziel;0
DialogDruckereinrichtung;0

D. h., es wird das Ausgabeziel aus der Druckvorlage genommen und keine Dialoge angezeigt, sodass sofort gedruckt wird.
Einschränkungen:
Mit dieser Funktion können keine Werkaufträge und keine Umlagerungsbelege (ab Wawi 10.1.6) gedruckt werden, da für diese innerhalb der Wawi ein Sonderhandling besteht. Werkaufträge und Umlagerungsbelege können über die Funktion XFakt.DruckeBeleg gedruckt werden. Bei der Angabe eines ungültigen Vorlagendateinamens wird immer die Standardvorlage für den entsprechenden Belegtyp genommen. Kann im Gegensatz dazu

	jedoch zu einer Vorlagenbezeichnung keine Druckvorlagendatei ermittelt werden, wird die Funktion mit Fehler beendet. **Rückgabewerte/Fehler:** Je nach Einstellung des Stillen Modus wird bei Fehlern entweder eine OleException generiert (Stiller Modus aus) oder die Funktion gibt einen Rückgabewert mit Fehlercode zurück (>0) und der Fehlercode wird zusätzlich in das Fehler-Objekt geschrieben. Dann kann er über LetzterFehler abgefragt werden (inkl. Fehlermeldung). Folgende Fehlercodes sind möglich: • 5 (OleException $8004 1005) ungültiger Optionsname, ungültiger oder fehlender Wert • 13 (OleException $8004 100D) ungültiger Übergabeparameter; z. B. nicht vorhandener Belegtyp, fehlende Belegnummer • 14 (OleException $8004 100E) ungültige Bezeichnung der Druckvorlage, d. h. es konnte kein Dateiname ermittelt werden • 15 (OleException $8004 100F) alle nicht weiter spezifizierten Fehler, z. B. Abbruch beim Email-"Druck" • 16 (OleException $8004 1010) Der Belegtyp ist vorhanden, wird durch diese Funktion jedoch zur Zeit nicht unterstützt. Gilt aktuell für die Belegtypen "W" (Werkauftrag) und „U“ (Umlagerungsbeleg, ab Wawi 10.1.6).
`Umlagern(Menge: Double; Datum: TDateTime; const Artikel: WideString; const VonLager: WideString; const InLager: WideString; const Mitarbeiter: WideString): OleVariant`	Lagert die angegebenen Menge (wenn vorhanden) VonLager InLager um. Die Werte für Mitarbeiter und Datum sind optional. Ab Wawi 9.0.4 werden auch Läger mit Lagerplätzen unterstützt. • Ab Wawi 10.1.2 kann auch aus Wareneingangslägern umgelagert werden. • Ab Wawi 10.1.6 entspricht das Verhalten dem der manuellen Umlagerung (Artikelstammdaten, Seite Lager), d. h. es wird kein Umlagerungsbeleg erstellt. Bis einschließlich Wawi 10.1.5 wurde ein Umlagerungsbeleg erstellt, der jedoch keine Positionen enthielt, da sich die Belegnummern von Belegkopf (Tabelle BELEG) und Umlagerungspositionen (Tabelle BELEGP) unterschieden. • Ab Wawi 11.0.1 kann statt der Artikelnummer auch eine Serien- oder Chargennummer angegeben werden. • In den Versionen 12.0 bis 12.0.1 war die Funktionalität abgeklemmt. • Ab Version 12.0.2 ist die Funktion `XFakt.Umlagern` an die neuen Lagerungsstrukturen angepaßt. Bei jedem Aufruf wird zur Zeit eine neue manuelle Lagerung (kompletter Beleg mit Belegkopf!) erstellt. Bis einschließlich der Versionen 11.5.x wurden nur Belegpositionen ohne echtem Belegkopf erzeugt.

`BildExport(Directory: String): Variant`	Die Funktion speichert Artikelbilder in den Ordner, der in Directory übergeben wird. Es werden nur Bilder von Artikeln mit gesetztem Feld "ShopAktiv" gespeichert (Filter). Da die Funktion schon älter ist, hat sie folgende Einschränkungen: • Pro Artikel wird nur ein Bild gespeichert - das mit der kleinsten "Ordnung" 0. • BMP-Bilder bekommen die Dateiendung .bmp. _Alle_ anderen Bildformate (tiff, gif, jpg, png, usw.) sind dem Export unbekannt und bekommen automatisch die Endung .jpg! D.h. der Export geht davon aus, dass es sich bei Nicht-bmp-Bildern um jpg-Bilder handelt.
`BilderExport(const BlobKey: WideString; const Key: WideString; Level: Integer; const Directory: WideString; const Filename: WideString): Integer;`	Diese Funktion stellt eine allgemeine Möglichkeit zum Export von Bildern aus der Bildtabelle zur Verfügung. Die Parameter Directory und Filename werden als Formel angegeben! Vor dem Speichern jedes Bildes werden nicht vorhandene Verzeichnisse angelegt. Außerdem wird das Dateidatum des Bildes auf den Wert aus der Tabellenspalte BILD.Datum gesetzt. Damit ist z. B. ein Datenabgleich mit Web-Shops anhand dieses Datums möglich, bei dem nur geänderte Bilddateien übertragen werden. (Das Datum kann im Bilderdialog innerhalb der Warenwirtschaft geändert werden.) Der Rückgabewert ist die Anzahl exportierter Bilder bzw. <0 bei einem Fehler. Diese Funktion unterstützt außerdem den Stillen Modus. Zum Testen der Übergabeparameter, Formelfunktionen und Optionen (Stiller Modus) kann das aktuelle Delphi-Demo mit einem Mustermandanten benutzt werden. **Bedeutung der Übergabeparameter:** • *BlobKey: WideString* Durch Angabe des BlobKeys ist eine Einschränkung auf Bilder bestimmter Stammdaten möglich. Intern handelt es sich um die ersten beiden Zeichen der Tabellenspalte BILD.Blobkey. Wird ein Leerstring übergeben, werden alle Bilder exportiert. • *Key: WideString* Durch Angabe des Stammdatenschlüssels ist eine Einschränkung auf die Bilder eines Stammdatensatzes möglich. Bei Übergabe eines Leerstrings erfolgt keine Einschränkung. Die Angabe von SQL-Platzhaltern (%, _) ist möglich. • *Level: Integer* Pro Stammdatensatz sind mehrere Bilder möglich. Die Reihenfolge entspricht der Reihenfolge im Bilderdialog. Das Bild mit Level=0 ist das Hauptbild. Ist Level<0, werden alle Bilder des Stammdatensatzes exportiert.

- *Directory: WideString*
  Verzeichnis, in dem die Bilder abgelegt werden sollen. Die Angabe erfolgt als Formel!
- *Filename: WideString*
  Dateiname, wobei ebenfalls die Angabe als Formel erfolgt!

mögliche Formelplatzhalter:

- *{BlobKey}*
  BlobKey der aktuellen Bilddatei (2 Zeichen),
  z. B.: AR = Artikelbild, MW = Mitarbeiter
- *{Key}*
  Schlüssel des Stammdatensatzes,
  z. B. 200001 = Artikel, 1 = Mitarbeiter
- *{Level}*
  Ordnungszahl des aktuellen Bildes:
  0 = Hauptbild, 1 = zweites Bild, usw.
- *{Ext}*
  Dateierweiterung des aktuellen Bildes ohne Punkt. Es werden dieselben Bildformate unterstützt, die auch im Bilderdialog angezeigt werden können.
  Standard sind BMP, GIF, JPG, PNG, TIF.
- *{Mandant xxx}*
  sämtliche Platzhalter für mandantenspezifische Daten,
  z.B.:

{Mandant Name}	Mandantenname
{Mandant Mandant}	Mandantenkürzel, z. B. WAWI, WUNDF

Setzen des Dateidatums:

Beim Setzen des Dateidatums anhand des Feldes BILD.Datum ist zu beachten, dass die Microsoft Zeitrechnung für das Dateidatum aus historischen Gründen erst am 1.1.1980 beginnt. Datumsangaben die vor diesem Datum liegen erzeugen eine OleException! NULL-Werte sind dagegen möglich: In diesem Fall wird das Datum nicht geändert und es bleibt das aktuelle Speicherdatum bestehen.

Rückgabewerte/Fehler:

Je nach Einstellung des Stillen Modus wird bei Fehlern entweder eine OleException generiert (Stiller Modus aus) oder die Funktion gibt einen Rückgabewert mit Fehlercode zurück (<0) und der Fehlercode wird zusätzlich in das Fehler-Objekt geschrieben. Dann kann er über `LetzterFehler` abgefragt werden (inkl. Fehlermeldung).

Für den Rückgabe-Fehlercode gilt folgende Beziehung:
Rückgabewert = - Fehlercode

Beim Fehlercode der OleException haben wir uns an die Microsoft-

Vorgaben gehalten. Der allgemeine OLE-Fehlercode ist $8004 0000 + Wawi-Offset von $1000 + Fehlercode.

Wie sich der "echte" Wawi-Fehler ergibt, kann anhand des Quellcodes des Delphi-Demos sowie der folgenden Auflistung nachvollzogen werden:

Fehlercodes

6 Parser-Fehler: Fehler in der Formelfunktion für Verzeichnis oder Dateiname, z. B. vergessene doppelte Anführungszeichen bei Konstanten
7 Fehler beim Anlegen des Verzeichnisses: "berechnetes" Verzeichnis konnte nicht erstellt werden
8 Fehler beim Speichern der Bilddatei: z. B. ungültige Zeichen im Dateinamen (*, ?), schreibgeschützte Datei existiert bereits
9 allg. Fehler: alle nicht weiter spezifizierten Fehler
12 Fehler beim Setzen des Dateidatums: z. B. BILD.Datum < 1.1.1980

```
BilderExportEx(const BlobKey,
Key: WideString; Level: Integer;
const FileTypes, Filter:
WideString; const Directory,
Filename: WideString; FileDate:
Integer): Integer;
```

Diese Funktion ist eine Erweiterung der Funktion `BilderExport()`. Alle dort gemachten Angaben gelten auch für diese Funktion.

Parameter:

- *FileTypes: WideString*
  In diesem Parameter kann eine durch Semikolons getrennte Auflistung von Grafik-Dateitypen für den Export angegeben werden. Der Parameter stellt eine „whitelist" dar - alle Bilder mit hier aufgeführten Dateiformaten werden exportiert.Wird ein Leerstring übergeben, werden alle Bilder exportiert.
  Bsp.: gif;bmp;png
  Intern erfolgt die Filterung nach dem Dateityp erst nach dem kompletten Laden eines Bildes aus der BILD-Tabelle und der Analyse des Grafikformates. Den Bildexport auf bestimmte Dateitypen zu beschränken ist deshalb eine relativ zeitaufwendige Sache. Über die restlichen Filtermöglichkeiten (BlobKey, Key, Level, Filter) sollte die Datenmenge bestmöglich eingegrenzt werden.

- *Filter: WideString*
  Hier kann ein Teil eines gültigen SQL-WHERE-Befehls für die interne SELECT..FROM..WHERE–Anweisung auf die BILD-Tabelle angegeben werden. Damit ist eine flexible Filterung von Datensätzen möglich.
  Der Filterstring wird nach folgendem Muster angehängt:
  SELECT <Felder> FROM BILD
  WHERE <Filter nach BlobKey/Key/Ordnung>AND <übergebener Filterstring>
  Bsp.: Kennzeichen1 = 1(Im Bilderdialog Checkbox „Export".)

	• *FileDate: Integer* Über den Parameter FileDate kann bestimmt werden, welches Dateidatum in der erzeugten Bilddatei gesetzt werden soll. (Zum Vergleich: Die Funktion BilderExport() setzte das Dateidatum immer auf das Datum der Tabellenspalte BILD.Datum.)Folgende Werte sind möglich: 0 nicht ändern (->Speicherdatum) 1 BILD.Datum (wie `BilderExport()`) 2 BILD.AngelegtAm 3 BILD.BearbeitetAm **Weitere Änderungen im Vergleich zu `BilderExport()`:** • Der Formelplatzhalter {Ordnung} ist neu und kann jetzt statt {Level} angegeben werden. Es handelt sich hierbei um ein Synonym, der Platzhaltername ist damit identisch zum Tabellenspaltennamen. • neuer Fehlercode: 17 Ungültiges SQL im Parameter Filter
`Listenpreis(Artikel: WideString): Variant`	Liefert den Listenpreis für den übergebenen Artikel zurück.
`ArtikelUmbuchen(const AlterArtikel: WideString; const NeuerArtikel: WideString; Datensicherung: WordBool; Statistikaufbau: WordBool; Meldung: WordBool): OleVariant`	Bucht einen Artikel um. **Parameter:** • *Datensicherung* legt fest, ob vorher eine Sicherung durchgeführt werden soll (empfiehlt sich in einer Schleife für den ersten Eintrag). • *Statistikaufbau* legt fest, ob anschließend die Statistik aufgebaut werden soll (empfiehlt sich in einer Schleife für den letzten Eintrag). • *Meldung* legt fest, ob im Erfolgsfall nach der Umbuchung eine Meldung ausgegeben werden soll.
`Mandant: Variant`	Liefert den aktuellen Mandanten.

`Import(const Muster: WideString; const ImportDatei: WideString): OleVariant`	Mit dieser Funktion können Importmuster, die mit dem Importassistenten erzeugt wurden, ausgeführt werden. In *Muster* wird die Bezeichnung und in *ImportDatei* der vollständige Dateiname übergeben, in der sich das Muster befindet. Muster können sich in der Import.Ini oder in Dateien, die mit dem Assistenten ausgeschleust wurden befinden. BDE- und SQL-Version unterscheiden sich, wobei die BDE-Version flexibler ist. **BDE:** Hier kann, wie oben beschrieben, ein beliebiger Dateipfad und -name für den Import-Dateinamen angegeben werden. **SQL:** In der SQL-Version kann nur ein bereits im Programm vorhandenes Muster verwendet werden, da die INI-Abfrage immer über die SQL-Server-Tabelle SL_DATEN.INIFILES läuft. Der Parameter *ImportDatei* muss immer IMPORT.INI sein.
`GetWerkauftrag(BelegNummer: WideString): Werkauftrag`	Erzeugt ein Objekt vom Typ Werkauftrag und lädt die Daten entsprechend der übergebenen Belegnummer. Die Funktionen des Objektes Werkauftrag werden unter Punkt 6 beschrieben.
`WerkauftragAnlegen(Stueckliste: String; Menge: Double): Variant`	Legt einen Werkauftrag an, lädt eine Stückliste mit der übergebenen Menge und gibt den Werkauftrag als Objekt zurück. Für weitere enthaltene Stücklisten mit Dispositionsart Auftrag, ohne Auflösung werden automatisch Werkaufträge angelegt, wenn die entsprechende Option in den Mandanteneinstellungen gesetzt ist.
`Benutzer: WideString`	Gibt das aktuelle Benutzerkürzel zurück.
`JahrMonat: WideString`	Die Funktion liefert die aktuelle Kombination Jahr/Monat im Format "JJJJMM".
`LoginReady: OleVariant`	
`SetLogonOk(SetOn: WordBool)`	
`CheckStatistik(MitAbfrage: WordBool)`	
`CreateOleAdress: IUnknown`	ab Wawi 11.0 nicht mehr unterstützt; reserviert für Cobra CRM Plus 2008-Provider
`GetBelegTypen: BelegTypen`	Liefert eine Liste aller vorhandenen Belegtypen zurück.
`ArtikelBestand(const Artikel: WideString; MitReserve: WordBool): OleVariant`	
`CreateLager: Lager`	

`WandelBelegEx(const VonBelegTyp: WideString; const VonBelegNummer: WideString; const NachBelegTyp: WideString; DialogAnzeigen: WordBool): WideString`	Belegübergabe. Liefert die neue Belegnummer. Mit *DialogAnzeigen* kann eingestellt werden, ob der Übergabedialog angezeigt werden soll oder nicht.

`WandelBelegEx2(const VonBelegTyp: WideString; const VonBelegNummer: WideString; const NachBelegTyp: WideString; DialogAnzeigen: WordBool; Mengenanpassung:WideString): WideString`	Arbeitet wie `WandelBelegEx`. Zusätzlich kann eine Liste (String mit Umbrüchen) für die Mengenanpassung der Positionen übergeben werden. Als letzte Spalte der Liste kann optional eine Seriennummer angehängt werden. Die einzelnen Bestandteile werden durch Semikolon getrennt.
	
	Format einer Zeile mit Mengenanpassung:
	
	`Quellbelegnummer;Positionsnummer; Artikelnummer;Menge`
	
	bzw.
	
	`Quellbelegnummer;Positionsnummer; Artikelnummer;Menge;Seriennummer`
	
	Ab Wawi 8.2.11 können statt der Seriennummern auch Chargennummern verwendet werden. Die Funktion sucht intern zunächst nach einer passenden Seriennummer - wird keine gefunden, wird in den Chargennummern gesucht:
	
	`Quellbelegnummer;Positionsnummer; Artikelnummer;Menge;Serien-/Chargennummer`
	
	Ab Wawi 9.0.3 ist optional die Angabe von Lager und Lagerplatz in der allgemein verwendeten Syntax `Lager,Dimension1,Dimension2,Dimension3` möglich:
	
	`Quellbelegnummer;Positionsnummer; Artikelnummer;Menge;Serien-/Chargennummer; Ziellager und Lagerplatz`
	
	Ab Wawi 12.0.2.8 ist die Angabe von Verfallsdatum und Preismenge möglich.
	
	Eine vollständige Zeile hat deshalb folgendes Format:
	
	`Quellbelegnummer;Positionsnummer; Artikelnummer;Menge;Serien-/Chargennummer;Ziellager,Lagerplatz; Verfallsdatum;Preismenge`
	
	Werden Lager und Lagerplatz angegeben, muß bei Nicht-Seriennummernartikeln der Abschnitt Serien-/Chargennummer leer

	bleiben: `A0001;1;Artikel0815;100;;Hauptlager,3,4`
`SetzeOptionen(Optionen: WideString)`	Diese Funktion dient dazu, unterschiedliche Betriebsarten einzustellen, bzw. weitere Parameter zur Steuerung zu übergeben. Der Parameter ist als Liste (String mit Umbrüchen) aufgebaut. Je nach Vorhaben können dann auch mehrere Parameter übergeben werden. Die Funktion ist somit erweiterbar. Das Setzen der Optionen wirkt sich nur temporär auf die aktuelle COM-Verbindung im aktuellen Mandanten aus. Die Option wird nicht dauerhaft umgestellt. Nach einem Mandantenwechsel wird die Option neu aus den INI-Einstellungen gelesen und muß bei Bedarf wieder gesetzt werden. Ab Wawi V. 10.1.2 unterstützt diese Funktion selbst den "Stillen Modus". Ab Wawi V. 12.0 stehen die Mandanteneinstellungen "automatisch lagern, wenn Lager vorgegeben" und "Auslagerungsvorschlag über alle Läger" nicht mehr zur Verfügung. Diese wurden auf die am besten passenden neuen Mandanteneinstellungen umgestellt (s. Tabelle).
Parameter für SetzeOptionen:	
`Stiller Modus`	Ein: `'Stiller Modus;1'` Aus: `'Stiller Modus;0'` In dieser Betriebsart werden keine Meldungen, Hinweise und Fragen (Messageboxen) vom Programm ausgegeben. Fragen werden intern immer mit ja beantwortet. Zusätzlich zu Messageboxen werden nur der Kreditlimithinweis und die Artikel-Lagerinformation unterdrückt. Um auf Meldungen des Programms weiterhin reagieren zu können, empfiehlt es sich, die Funktion `LetzterFehler` aufzurufen.
`Beleg.Lagerung.Einlagerungsdialog.ImmerAnzeigen`	Ein- und Ausschalten der Mandantenoption "Seite Lager | Einlagern | Lagerdialog immer anzeigen"
`Beleg.Lagerung.Einlagerungsdialog.SCBearbeitungsdialogAnzeigen`	Ein- und Ausschalten der Mandantenoption "Seite Lager | Einlagern | Nach der Erfassung von S/C-Nr. Bearbeitungsdialog anzeigen"
`Beleg.Lagerung.Auslagerungsdialog.ImmerAnzeigen`	Ein- und Ausschalten der Mandantenoption "Seite Lager | Auslagern | Lagerdialog immer anzeigen"

`Beleg.Lagerung.Auslagerungsdialog.AutomatischLagern`	Ein- und Ausschalten der Mandantenoption "Seite Lager | Auslagern | Automatisch, wenn Lager vorgegeben" ***Achtung:*** *Ab Wawi V. 12.0 ist diese Mandanteneinstellung nicht mehr verfügbar. Mit dieser COM-Option wird die Mandanteneinstellung "negativ lagern" ein- und ausgeschaltet.*
`Beleg.Lagerung.Auslagerungsdialog.VorschlagAlleLaeger`	Ein- und Ausschalten der Mandantenoption "Seite Lager | Auslagern | Vorschlagsliste über alle Läger erstellen" ***Achtung:*** *Ab Wawi V. 12.0 ist diese Mandanteneinstellung nicht mehr verfügbar. Mit dieser COM-Option wird die Mandanteneinstellung "Bestände vorschlagen aus" auf "Positionslager, dann alle Läger" (eingeschaltet = 1), sonst auf "Positionslager" (ausgeschaltet = 0) eingestellt.*
`Beleg.U.Druck.Sortierung`	Unterdrückung des Sortierreihenfolge-Dialoges beim Druck von Umlagerungsbelegen. Gültige Werte sind die Sortiereinstellungen im Dialog: • Artikelnummer, Bezeichnung • Positionserfassung • Leerstring = Löschen der Einstellung Die Optionen sind nicht sprachenabhängig, d. h. auch mit englischem Frontend können die deutschen Bezeichnungen verwendet werden, da diese intern übersetzt werden. **Parameter:** • *Leerstring* (Löschen der Einstellung) Anzeige des Sortierreihenfolge-Dialoges (altes Verhalten bis Wawi 10.1.6 sowie aktuelles Standardverhalten) • *ungültige Einstellung* Es wird die erste Sortieroption des Dialoges eingestellt (z. Zt. Artikelnummer, Bezeichnung)
`System.Version.Hauptversion` `System.Version.Unterversion1` `System.Version.Unterversion2` `System.Version.Unterversion3`	Rückgabe des entsprechenden Teils der Versionsnummer. Diese Werte können nur gelesen werden! Bsp.: Wawi-Version: 11.1.2.3 *Rückgabe*: (Datentyp String!) *Hauptversion*: 11 *Unterversion1*: 1 *Unterversion2*: 2 *Unterversion3*: 3 Da `XFakt.LiesOptionen` eine String-Liste verarbeiten kann, können alle vier Teile mit einem Aufruf abgefragt werden.

`LetzterFehler: Fehler`	LetzterFehler kann im ‚Stillen Modus' (Siehe Funktion `SetzeOptionen`) aufgerufen werden und gibt einen Record mit der Fehlerbeschreibung zurück. Beschreibung des Datentyps Fehler in Punkt 12 (Datentypen)
`procedure StartMakro(const MakroDatei: WideString);`	Diese Funktion führt die angegebene Makro-Datei aus. Die Makro-Datei muß sich im Verzeichnis \Makro der Installation befinden. Der Dateiname muß komplett mit Erweiterung angegeben werden. Enthält das auszuführende Makro Parameter, so werden diese Werte wie bei einem Aufruf innerhalb der Wawi über Dialoge abgefragt. Beispiel: `Fakt.StartMakro('test.qdf');`
`function CreateMakroParamList: MakroParamList;`	Diese Funktion gibt ein Makro-Parameterlisten-Objekt zur Übergabe von Makro-Parametern und ihren Werten an die Funktion `StartMakroEx` zurück.

`procedure StartMakroEx(const MakroDatei: WideString; const ParamList: MakroParamList);`	Diese Funktion führt die angegebene Makro-Datei aus. *ParamList* ist ein zuvor über `CreateMakroParamList` angefordertes Objekt und enthält die Parameternamen und ihre Werte.Die Makro-Datei muß sich im Verzeichnis \Makro der Installation befinden. Der Dateiname muß komplett mit Erweiterung angegeben werden. Es ist jedoch nicht notwendig, die Makro-Datei im SelectLine-Programm in den Makro-Assistenten einzuschleusen.String-Parameter-Werte sollten in doppelte Anführungszeichen eingeschlossen werden, damit diese korrekt vom Programm erkannt werden. Wenn der String-Parameter-Wert Leerzeichen enthält, ist dies Pflicht!
	
	Beispiel:
	
	`var`
	`makroParamList: OleVariant;`
	`fakt: OleVariant; { Voraussetzung: Die Variable fakt ist bereits initialisiert... }`
	`makroParamList := Fakt.CreateMakroParamList;`
	`makroParamList.AddParam('Param1', 'Param1_Wert');`
	`makroParamList.AddParam('Param2', 'Param2_Wert');`
	`{ usw. }`
	`Fakt.StartMakroEx(`
	`'MakroDatei.qdf', makroParamList`
	`);`
	`makroParamList := Unassigned;`

`property MandantLand: COMMandantLand;`	Abfrage des Mandantenlandes. Der Rückgabewert ist vom Typ Integer mit folgenden definierten Werten: 0 = cmlSchweiz 1 = cmlDeutschland 2 = cmlAustria
`property SQLServer: WideString;`	Rückgabe der verbundenen SQL-Server-Instanz. Die BDE-Version liefert eine Exception mit Fehlermeldung zurück.

`procedure KontaktadresseAnzeigen(const AdressBlobKey: WideString; KontaktID: Integer); function DatensatzAnlegen(const BlobKey: WideString; var Nummer: WideString; FieldList: OleVariant; ValueList: OleVariant): WordBool; function KontaktadresseAnlegen(const HauptBlobKey: WideString; const HauptNummer: WideString; var AdressID: WideString; FieldList: OleVariant; ValueList: OleVariant): WordBool;`	Diese Funktionen sind reserviert und werden von der Warenwirtschaft nur intern zum Zugriff auf den Cobra CRM Plus 2008-Provider benutzt.
`function HatLizenz(aLizenzOption: LizenzOption): WideString;`	Abfrage einer Lizenzausprägung. Wenn die Lizenz vorhanden ist, wird „J“ zurückgegeben, sonst „N“. Folgende Lizenzen können abgefragt werden: 0 = lizenzDemo 1 = lizenzStandard 2 = lizenzStandardPlus 3 = lizenzGold 4 = lizenzPlatin 5 = lizenzDiamond 6 = lizenzMaskenEditor 7 = lizenzToolboxRuntime 8 = lizenzToolboxEdit 9 = lizenzCobraErpProvider 10 = lizenzSprache 11 = lizenzEasy 12 = lizenzStampit (wird ab Wawi 12.0 nicht mehr unterstützt) 13 = lizenzWawiStandort 14 = lizenzWawiIntra 15 = lizenzWawiMosaic 16 = lizenzWawiMapkit 17 = lizenzWawiXtrade 18 = lizenzReweFibu 19 = lizenzReweKonas 20 = lizenzReweKonsul 21 = lizenzReweChOPOS 22 = lizenzReweAnlag 23 = lizenzReweKost 24 = lizenzLohnKUG 25 = lizenzLohnFlexi 26 = lizenzLohnZVK 27 = lizenzLohnPfaendung 28 = lizenzLohnDakota 29 = lizenzLohnBau 30 = lizenzVertriebsModul

`function LiesOptionen(var Optionen: WideString): WordBool;`	Diese Funktion liest aktuell gesetzte Optionen aus. Falls diese nicht über die Funktion `SetzeOptionen` geändert wurden, handelt es sich um die aktuell gültigen Mandanteneinstellungen. Die Optionen gelten nur temporär für die aktuelle COM-Verbindung im aktuellen Mandanten. Sie können sich also sowohl von anderen gleichzeitig gestarteten COM-Programmen als auch von den Mandanteneinstellungen der interaktiv arbeitenden Benutzer unterscheiden. Es sind alle über SetzeOptionen änderbaren Optionen angebbar. Groß-und Kleinschreibung wird nicht unterschieden. Da es sich um eine Liste handelt, können auch mehrere Optionen gleichzeitig abgefragt werden, wobei Leerzeilen ignoriert werden. Der Rückgabewert ist False (=0) bei einem Fehler, sonst True (<>0). Diese Funktion unterstützt den "Stillen Modus", d. h. Fehler können über den Rückgabewert und `LetzterFehler` abgefragt werden (Stiller Modus ein) oder es wird eine Exception generiert (Stiller Modus aus). ***Achtung*** *Der Parameter Optionen wird in jedem Fall geändert: Entweder werden die aktuellen Einstellungen zurückgegeben (Format: Option;Wert) oder es wird im Fehlerfall (->ungültige Option) die erste ungültige Zeile mit einem Fehlertext beschrieben.*

`function EMailKontaktAnlegen(IdAdresse: Integer; const EMailAdresse: WideString; const Betreff: WideString; const Richtung: WideString; const Text: WideString; IdStatus: Integer; PrivatKontakt: Integer; const Pfad: WideString; Datum: TDateTime; Automatik: WordBool): WideString;`
 	
`function WaehleAdresse(const EMailAdresse: WideString; out DialogResultOk: WordBool): IAdressList;`
`function KontaktStatusListe: KontaktStatusList;`
`function SucheAdresse(const EMailAdresse: WideString): IAdressList;`
	Diese Funktionen sind reserviert und werden von der Warenwirtschaft nur intern zum Zugriff auf das SelectLine Outlook-Add-In benutzt.

<table>
<tr><td><code>function SetzeArtikelInaktivStatus(Status : WordBool; const Artikelnummer: WideString; Stueckliste: WordBool; Alternativ: WordBool; Zubehoer: WordBool; Zuschlag: WordBool; Referenzen: WordBool; Immer: WordBool; Fehlerprotokoll: WordBool; ComZugriff: WordBool): WordBool;</code></td><td>Die Funktion setzt den Inaktiv-Status des Artikels. Der Stille Modus wird unterstützt.

Parameter:
• Status
<code>False</code> – Artikel aktiv setzen
<code>True</code> – Artikel inaktiv setzen

• Stueckliste, Alternativ, Zubehoer, Zuschlag, Referenzen
<code>True</code> – Artikel aus Stücklisten/als Alternativartikel/als Zubehör/ als Zuschlagartikel/ alle Referenzen löschen

• Immer
<code>True</code> – Artikel trotz "Fehler" inaktiv setzen

• Fehlerprotokoll
<code>True</code> – Anzeige eines Fehlerprotokolls

• ComZugriff
<code>True</code> – Unterdrückung zusätzlicher Erfolgs-/Fehlerdialoge</td></tr>
<tr><td><code>function LeseArtikelInaktivStatus(const Artikelnummer: WideString): SYSINT;</code></td><td>Die Funktion liest den Inaktiv-Status des Artikels. Der Stille Modus wird unterstützt.

Rückgabewerte:
<table><tr><th>Status</th><th>Rückgabewert Stiller Modus an</th><th>Rückgabewert Stiller Modus aus</th></tr><tr><td>Artikel nicht vorhanden</td><td>-1</td><td>OleException $800410129 (-2147217390d)</td></tr><tr><td>Artikel ist nicht inaktiv</td><td>0</td><td>0</td></tr><tr><td>Artikel ist inaktiv</td><td>1</td><td>1</td></tr><tr><td>Rückgabewert bei Exceptions</td><td>9 (s. XFakt.LetzterFehler)</td><td>wird direkt durchgereicht</td></tr></table></td></tr>
<tr><td><code>function SucheAdresseUeberId(Id: Integer): IAdressList;</code></td><td>Diese Funktionen ist reserviert und wird von der Warenwirtschaft nur intern zum Zugriff auf das SelectLine Outlook-Add-In benutzt.</td></tr>
<tr><td><code>function LoginIntern(const Schluessel: WideString): WordBool;</code></td><td>Reservierte Funktionen für die SelectLine Rewe-Wawi-COM-Kopplung.</td></tr>
<tr><td><code>function LogoutIntern(const Schluessel: WideString): WordBool;</code></td><td>Reservierte Funktionen für die SelectLine Rewe-Wawi-COM-Kopplung.</td></tr>
</table>

4 Tabellen-Objekt - IOleTable

Die Erzeugung des Objekts erfolgt mit der Funktion `CreateTable` des Fakt-Objektes.

Funktion	Beschreibung
`First`	Der Datensatzzeiger stellt sich auf die erste Position.
`Last`	Der Datensatzzeiger stellt sich auf die letzte Position.
`Next`	Der Datensatzzeiger stellt sich auf die nächste Position.
`Prior`	Der Datensatzzeiger stellt sich auf die vorherige Position.
`EoF: Boolean`	Diese Funktion liefert True, wenn der letzte Datensatz mit Next verlassen wurde oder kein Datensatz in der Tabelle existiert.
`BoF: Boolean`	Diese Funktion liefert True, wenn der erste Datensatz mit Prior verlassen wurde oder kein Datensatz in der Tabelle existiert.
`RecordCount: Long`	Diese Funktion liefert die Anzahl der Datensätze der Tabelle.
`GetValue(AFieldName: String; Var Value: Variant): Variant`	Der Inhalt des Feldes mit der Bezeichung *AFieldName* wird zurückgeliefert. Value enthält den Wert als Variante, der Rückgabetyp der Funktion ist der Datentyp des Feldes.
`PutField(AFieldName: String; Value: Variant): Boolean;`	Der Inhalt des Feldes mit der Bezeichung *AFieldName* wird mit dem Wert *Value* belegt. Hinweis: Die endgültige Speicherung von geänderten Werten erfolgt erst nach dem Aufruf der Funktion *Post*.
`Append`	Ein neuer Datensatz wird an die Tabelle angehängt und steht zur Bearbeitung bereit.
`Insert`	Ein neuer Datensatz wird in die Tabelle eingefügt und steht zur Bearbeitung bereit. Hinweis: Bei Tabellen, die einen Primärindex besitzen (zusätzlich gibt es eine Datei mit der Endung PX), hat die Funktion *Append* die gleiche Wirkung wie die Funktion *Insert*.
`Delete`	Der aktuelle Datensatz wird gelöscht.
`FindKey(Value: Variant): Variant`	Der Wert *Value* wird über den Primärindex der Tabelle gesucht. Wird er gefunden, wird der Datensatzeiger entsprechend eingestellt und die Funktion liefert True.
`FindKey2(Value1, Value2: Variant): Variant;`	
`FindKey3(Value1, Value2, Value3: Variant): Variant;`	

`FindKey4(Value1, Value2, Value3, Value4: Variant): Variant;`	
`Post`	
`PutValue(AFieldName: String; Value: Variant): Variant;`	Der Inhalt des Feldes mit der Bezeichung *AFieldName* wird mit dem Wert *Value* belegt. Hinweis: Die endgültige Speicherung von geänderten Werten erfolgt erst nach dem Aufruf der Funktion *Post*.
`GetField(AFieldName: String): Variant;`	Der Inhalt des Feldes mit der Bezeichung *AFieldName* wird zurückgeliefert.

`GetFields : Variant;`	Liefert Feld mit Namen, Datentyp und Größe der Tabellenfelder:
	
	`Array[3i+1] = Name`
	`Array[3i+2] = Datentyp`
	`Array[3i+3] = Size`
	`i=0,1...`

`GetIndex: WideString;`	Liefert den aktuell gesetzten Index
`SetIndex(anIndex : WideString);`	Setzt den Tabellenindex. Zurücksetzen auf den Primärindex der Tabelle: • BDE: Übergabe eines Leer-Strings '' • SQL: Übergabe eines Leer-Strings '' oder des Bezeichners 'PRIMARY'

Hinweis: *Die Rechte aus der Passwortverwaltung der Warenwirtschaft hinsichtlich der Datenzugriffe (sehen, ändern, anlegen, löschen) werden auch beim Zugriff über die COM-Schnittstelle überprüft. Ggf. werden entsprechende Hinweise angezeigt und der Zugriff unterbunden.*

4.1 Änderungen an OleTable ab Wawi V. 11.0

Mit der Version 11 von Warenwirtschaft und Rechnungswesen wurde das Verhalten der COM-Komponente OleTable überarbeitet und hinsichtlich SQL optimiert.

Bis einschließlich Version 10.1.x wurde beim Anfordern von Tabellen mit `OleTable.CreateTable()` die Datenbanktabelle komplett geöffnet und alle Datensätze mit `„select *“` sofort geholt. Selbst wenn später mit `FindKey()` nur bestimmte Datensätze gesucht wurden. Das Abfordern eigentlich überflüssiger Datensätze dauerte bei Tabellen mit vielen Datensätzen dementsprechend lange, teilweise bis in den Sekundenbereich hinein.

Ab der Version 11 wurde das Handling intern auf SQL-Zugriffe optimiert. Beim Anfordern eines OleTable-Objektes werden jetzt keine Datensätze mehr automatisch geholt. Erst bei weiteren Zugriffen werden nur die erforderlichen Daten abgerufen. Durch das neue Verhalten ergeben sich insbesondere bei großen Tabellen erhebliche Performancevorteile. Als Folge dieser Optimierungen hat sich das Verhalten von `OleTable.RecordCount()` minimal verändert.

4.1.1 Änderungen an OleTable.RecordCount()

Bis Version 10.1.x war der Wert über die komplette „Lebenszeit" der OleTable konstant (abzüglich/zuzüglich gelöschter/eingefügter Datensätze über die OleTable). Änderungen außerhalb der verwendeten OleTable (z. B. durch andere Benutzer) waren grundsätzlich nicht sichtbar.
Ab Version 11 wird `OleTable.RecordCount` immer aktuell mit „`select count(*)`" geholt. Der Wert ist damit nicht mehr konstant, da Änderungen von außerhalb der OleTable in die Zählung mit eingehen. Durch diese Änderung hat `RecordCount` nur noch "informativen Charakter". Um alle Datensätze zu durchlaufen, sollten feste Schleifen über RecordCount vermieden werden. Stattdessen sollten unbestimmte Schleifen (repeat/while) bis OleTable.EoF durchlaufen werden.

4.1.2 Programmtechnische Überlegungen zur Verwendung von OleTable-Objekten:

Wird über COM auf SelectLine SQL-Programme mit Versionsnummer 10.x und kleiner zugegriffen, so kann es bei Verwendung von OleTable-Objekten auf Tabellen mit vielen Datensätzen zu Performance-Einbrüchen kommen. (Die BDE-Versionen sind nicht betroffen.) Hier ist die einzige Lösung, das COM-Programm auf OleQuery umzustellen. (Was wiederum bei BDE-Versionen langsam sein kann) Bei SelectLine SQL-Programmen ab Version 11 bestehen performance-technisch praktisch keine Unterschiede zwischen OleTable und OleQuery. Dies bedeutet, dass COM-Clients durch Umstellung des Servers auf Version 11 schneller laufen, wenn OleTable-Objekte verwendet werden.

Die folgende Tabelle gibt einen Überblick über die verschiedenen Kombinationsmöglichkeiten:

SelectLine COM-Server	OleTable	OleQuery
BDE V. <= 10.x	schnell	schnell nur langsam, wenn auf nicht-indizierte Felder zugegriffen wird
SQL V. <= 10.x	langsam bei vielen Datensätzen (wg. `select *`) evtl. umstellen auf OleQuery, wenn COM-Client nie mit BDE-Versionen benutzt wird (da OleQuery dort evtl. langsam ist)	schnell Performance ist nur abhängig vom SQL Server
SQL V. >= 11.x	schnell aufgrund interner Optimierungen in OleTable COM-Clients, die OleTables verwenden, werden evtl. automatisch schneller	schnell Performance ist nur abhängig vom SQL Server

5 Query-Objekt - IOleQuery

Die Erzeugung des Objekts erfolgt mit der Funktion `CreateQuery()` des Fakt-Objektes.

Funktion	Beschreibung
SetSQLText(SQLText: string)	SQL-Statement setzen
OpenSQL	SQL-Statement anzeigen (z.B. select-Anweisung)
ExecuteSQL	SQL-Statement ausführen (z.B. update-, delete-, insert-Anweisungen)
CloseSQL	Cursor schließen
First	Der Datensatzzeiger stellt sich auf die erste Position.
Last	Der Datensatzzeiger stellt sich auf die letzte Position.
Next	Der Datensatzzeiger stellt sich auf die nächste Position.
Prior	Der Datensatzzeiger stellt sich auf die vorherige Position.
EoF: Boolean	Diese Funktion liefert True, wenn der letzte Datensatz mit Next verlassen wurde oder kein Datensatz in der Tabelle existiert.
BoF: Boolean	Diese Funktion liefert True, wenn der erste Datensatz mit Prior verlassen wurde oder kein Datensatz in der Tabelle existiert.
RecordCount: Long	Diese Funktion liefert die Anzahl der Datensätze der Tabelle.
GetValue(AFieldName: String; Var Value: Variant): Variant	Der Inhalt des Feldes mit der Bezeichung AFieldName wird zurückgeliefert. Value enthält den Wert als Variante, der Rückgabetyp der Funktion ist der Datentyp des Feldes.
GetField(AFieldName: String): Variant;	Der Inhalt des Feldes mit der Bezeichung AFieldName wird zurückgeliefert.

6 Beleg-Objekt - IOleBeleg

Mit dem Beleg-Objekt können Belege erstellt werden. Das Objekt wird mit der Funktion `NeuerBeleg()` des Fakt-Objektes erstellt. Alle Funktionen die mit "Set" beginnen haben dieselbe Funktionalität wie in der Belegmaske, wenn ein Wert in das entsprechende Feld eingegeben wird.
(z.B.: SetAdresse füllt alle Adressfelder etc. des Belegs mit den Vorgabewerten aus dem Kundenstamm)

Funktion	Bemerkung
SetDatum(const Datum: WideString)	Ändert oder setzt das Belegdatum.
SetAdresse(const KunLie: WideString)	Ändert oder setzt die Belegadresse. KunLie muss eine gültige Kunden bzw. Lieferantennummer enthalten.

SetMitarbeiter(const Mit: WideString)	
SetLiefertermin(Liefertermin: TDateTime)	
SetPreisgruppe(const Preisgruppe: WideString)	Gültige Werte sind '0'-'9'
SetPreistyp(const PreisTyp: WideString)	Ändert oder setzt den Preistyp. Gültige Werte sind 'B'-Brutto, 'N'-Netto, 'S'-Steuerfrei
SetRabattgruppe(const RabGru: WideString)	RabGru - gültiger Schlüssel aus den Rabattgruppen
SetBelegRabatt(Belegrabatt: double)	Prozentualer Beleg-Rabatt
SetWaehrung(const FWCode: WideString)	FWCode - gültiger Schlüssel aus den Währungen
SetWaehrungsfaktor(Waehrungsfaktor: Double)	Überschreibt Währungsfaktor.
SetBankbezug(const Bankbezug: WideString)	Bankbezug - gültiger Schlüssel aus den Bankbezügen
SetZahlungsbedingung(const ZahlBed: WideString)	ZahlBed - gültiger Schlüssel aus den Zahlungsbedingungen
SetLieferbedingung(const LiefBed: WideString)	LiefBed - gültiger Schlüssel aus den Lieferbedingungen
CreatePos(const Zeilentyp: WideString): OlePos	Erstellt ein Positionsobjekt. Gültige Zeilentypen sind A-Artikel, K-Kommentar, Z-Zwischensumme, T-Teilsumme, H-Handelstückliste, G-Unterartikel, W-Seitenwechsel, S-Gliederungssumme, E-Versand
CreateUnterPos(const Zeilentyp: WideString; AHauptkennung: WideString): OlePos	Erstellt ein Positionsobjekt, das einer Hauptposition zugeordnet werden kann, z.B. weitere Stücklistenartikel mit Zeilentyp G. Beispiel: MainPos := Rechnung.CreatePos('H'); MainPos.SetArtikel('1001'); MainPos.Save(1); { Parameter wird ab Wawi 10.0 ignoriert } Pos := Rechnung.CreateUnterPos('G', MainPos.GetKennung); Pos.SetArtikel('1002'); Pos.Save(2); { Parameter wird ab Wawi 10.0 ignoriert }
GetField(AFieldName: WideString): OleVariant	Der Inhalt des Feldes mit der Bezeichnung *AFieldName* wird zurückgeliefert.
PutField(const AFieldName: WideString; Value: OleVariant): OleVariant	Der Inhalt des Feldes mit der Bezeichnung *AFieldName* wird mit dem Wert *Value* belegt. Hinweis: Die endgültige Speicherung von geänderten Werten erfolgt erst nach dem Aufruf der Funktion *Save*. (Achtung! Einige Felder werden durch den Aufruf der Funktion *Save* neu berechnet.) Die "manuelle" Änderung von Feldern ist nur bei genauer Kenntnis deren Funktion ratsam!
AddFusstextLine(const Kommentar: WideString)	

AddKopftextLine(const Kommentar: WideString)	
SetRundung(const Rundung: WideString)	
EditPos(Kennung: WideString): OlePos	Liefert ein Positionsobjekt anhand seiner Kennung. Dieses muss zum aktuellen Belegsatz gehören.
EditPosIdx(Index: Integer): OlePos	Liefert ein Positionsobjekt anhand seiner Position im Beleg. 0-liefert die erste Position, 1-die zweite Position und so weiter.
GetAnzahlSteuer: Integer	Verwendung intern
GetSteuerBetrag(Index: Integer): Double	Verwendung intern
GetSteuerProzent(Index: Integer): Double	Verwendung intern
Save	Diese Funktion speichert den Beleg. Vor dem Speichern des Beleges wird der komplette Beleg wertmäßig neu kalkuliert. Dabei werden die folgenden Felder des Beleges neu berechnet (ohne Anspruch auf Vollständigkeit - Stand Wawi 10.0): • Brutto, Netto, Steuer • EuroBrutto, EuroNetto, EuroSteuer • FremdwaehrungBrutto, FremdwaehrungNetto, FremdwaehrungSteuer • Gewicht, Erloes, ErloesEuro, Status, UebernahmeOffen Diese Felder über *PutField* zu verändern ist damit wirkungslos, da sie immer überschrieben werden. Eine Möglichkeit, diese Felder nach dem Speichern des Beleges doch noch zu beschreiben, wäre es, über ein OleTable- bzw. OleQuery-Objekt die Werte direkt zu setzen (Mit den damit verbundenen eventuellen Dateninkonsistenzen).
DeletePos(Kennung: WideString): OleVariant	
PosKennung(Index: Integer): OleVariant	
SetIhrAuftrag(const AuftrNr: WideString)	
SetIhrAuftragVom(Auftragsdatum: TDateTime)	
Delete	

BucheSeriennummern(Artikelnummer, Seriennummern: WideString)	Die Funktion lagert entsprechend des Belegtypes mehrere Seriennummern. In der Verarbeitung ist die Anlage einer Belegposition, die Auswahl eines Artikels und das Speichern der Belegposition enthalten. Die Seriennummern sind bei der Übergabe als Liste (String mit Umbrüchen) zu übergeben. **Achtung!** In den Wawi-Versionen 12.0 bis 12.0.2.8 funktionierte die Auslagerung von S/C-Artikeln mit dieser COM-Funktion nicht. Der Fehler wurde mit Version 12.0.2.9 korrigiert.
property PositionsRundung: Integer;	Zugriff auf die Rundung der Belegpositionen (Bsp: 100 = 2-stellig, 1000 = 3-stellig, usw.)
property BelegRundung: Integer;	Zugriff auf die Rundung des Belegpreises (Bsp: 100 = 2-stellig, 1000 = 3-stellig, usw.)
SetStandort(const aStandort: WideString)	Setzt den Standort im Beleg. Anschließend können in den Belegpositionen nur Läger verwendet werden, denen dieser Standort zugeordnet ist.
property AutoAddZubehoer: WordBool;	• **False**: kein Zubehörhandling aktiv; Zubehör wird nicht beachtet • **True**: Das Zubehörhandling der Warenwirtschaft ist aktiv. Je nach Einstellung der Auswahlbox "Einfügemodus" im Dialog "Zubehör bearbeiten" des Artikelstammdatendialoges wird Zubehör automatisch eingefügt ("Immer") oder kann per Dialog ausgewählt werden ("Auf Nachfrage"). Die Einstellung "Manuell" zum manuellen Einfügen über das Extras-Menü des Beleges ("Zubehör einfügen" - ALT+Z) steht per COM nicht zur Verfügung.
property AutoAddZuschlag: WordBool;	• **False**: Zuschlagartikel werden nicht eingefügt • **True**: Zuschlagartikel werden automatisch eingefügt

property Lager: WideString;	Mit diesem Property kann das Beleglager gesetzt bzw. gelesen werden. Ein Setzen des Beleglagers über COM ist nur bei neu angelegten Belegen ohne Positionen sinnvoll, da sonst eine Abfrage zur Anpassung des Lagers der Positionen erfolgt. Vor dem Setzen des Lagers wird überprüft, ob das Lager gültig ist (passend zum Standort; Wareneingangslager bei Nicht-Lieferbelegen). Bei unpassendem Lager erscheint eine Fehlermeldung (Stiller Modus aus) bzw. es wird LetzterFehler gesetzt (Stiller Modus an).
	Beispiel:
	`Beleg := Fakt.NeuerBeleg('R');`
	`Beleg.SetStandort('2');`
	`Beleg.SetLager('202');`
	`<Positionen anlegen>`
	`Beleg.Save;`

7 Positions-Objekt - IOlePos

Mit dem Positions-Objekt können Belegpositionen erstellt werden. Das Objekt wird mit der Funktion `CreatePos()` des Beleg-Objektes erstellt. Alle Funktionen die mit "Set" beginnen haben dieselbe Funktionalität wie in der Positionsmaske, wenn ein Wert in das entsprechende Feld eingegeben wird. (z.B.: `SetArtikel` füllt alle Felder der Position mit den Vorgabewerten aus dem Artikelstamm) Folgende Funktionen stehen im Positionsobjekt zur Verfügung:

Funktion	Beschreibung
`SetEingabeMenge(const EingabeMenge: WideString)`	Achtung Eingabemenge ist ein String und kann auch Formeln enthalten (z.B.: 2 * 3)
`SetArtikel(Artikel: WideString)`	Artikel - gültiger Schlüssel aus den Artikelstammdaten oder EAN-Nummer bzw. Serien-/Chargennummer einer gültigen Artikelnummer. ***Achtung:*** *In den Wawi-Versionen 12.0 bis 12.0.2.8 funktionierte die Auslagerung von S/C-Artikeln mit dieser COM-Funktion nicht. Der Fehler wurde mit Version 12.0.2.9 korrigiert.*
`SetEinzelPreis(EinzelPreis: Double)`	Setzt den Einzelpreis der Belegposition. Die Mandanteneinstellung für die Rundung der Artikelpreise (Mandant - Einstellungen, Seite Artikel - Option Preise: Anzahl Nachkommastellen) wird beachtet. Es ist also nicht möglich, hiermit genauere Werte als bei interaktiver Belegbearbeitung in die Position zu schreiben.
`SetRabatt(Rabatt: Double)`	
`SetGesamtPreis(GesamtPreis: Double)`	
`SetVertreter(const Vertreter: WideString)`	
`SetKostenstelle(Kostenstelle: WideString)`	
`SetLager(Lager: WideString)`	Ab der Wawi V.9.0.3: Für Läger mit Lagerplätzen (Freies Lager, Flächenlager, Regallager) ist folgendes Format zu verwenden: `NNN,D1,D2,D3` *NNN:* Schlüsselnummer Stammdaten Lager *D1 – D3:* Dimensionen durch Komma getrennt. Beispiel Freies Lager: `100,1` Beispiel Regallager : `110,3,5,8` Für Lager ohne Lagerplätze wird, wie in der Version 8, die Schlüsselnummer des Lagers übergeben.

`SetBezeichnung(Bezeichnung: WideString)`	
`SetZusatz(Zusatz: WideString)`	
`SetFibukonto(Fibukonto: WideString)`	
`SetPreisEinheit(PreisEinheit: Double)`	
`SetMengenEinheit(Mengeneinheit: WideString)`	Mengeneinheit - gültiger Schlüssel
`SetSteuerCode(SteuerCode: WideString)`	SteuerCode - gültiger Schlüssel
`SetSteuerProzent(SteuerProzent: Double)`	Überschreibt die Steuerprozente
`Save(Level: Integer): WideString`	Level - Ebene (z.B. 1). ***Hinweis:*** *Das Objekt muß anschließend freigegeben werden.* **Änderungen:** • ab Wawi 12.0: Der Rückgabewert hat den Datentyp WideString statt Integer. • ab Wawi 10.0: Der Parameter Level wird ignoriert. Es kann ein beliebiger Wert übergeben werden. Das Erzeugen von Unterpositionen ist nur noch über den Parameter AHauptkennung der Funktion `IOleBeleg.CreateUnterPos()` möglich.
`AddKommentarLine(const Kommentar: WideString)`	
`SetFreiesFeld(AFieldname: WideString; Value: OleVariant)`	Der Inhalt eines der Freifelder mit der Bezeichung *AFieldName* (Frei1, Frei2, FreiDatum, FreiZahl) wird mit dem Wert *Value* belegt.
`PutExtraField(const AFieldName: WideString; Value: OleVariant): OleVariant`	Der Inhalt des Extrafeldes mit der Bezeichung *AFieldName* wird mit dem Wert *Value* belegt.
`GetExtraField(const AFieldName: WideString): OleVariant`	Gibt den Inhalt des Extrafeldes mit der Bezeichnung *AFieldName* zurück
`property Termin: TDateTime`	Setzt bzw. liest das Feld „Termin“ der Belegposition
`property Menge: Double`	Setzt bzw. liest das Feld „Menge“ der Belegposition
`UmrechnungEinzelpreis(const Artikel: WideString; Preis: Double; const PreisMengenEinheit: WideString; const ZielMengeneinheit: WideString): Double`	Verwendung intern; für SQL nicht implementiert!
`GetKennung: WideString`	Gibt den Wert des Feldes Kennung zurück (->GUID).
`property Artikel: WideString`	

```
property Bezeichnung:
WideString
property Zusatz: WideString
property Lager: WideString
property Vertreter: WideString
property Fibukonto: WideString
property Kostenstelle:
WideString
property SteuerCode: WideString
property Mengeneinheit:
WideString
property Postext: WideString
property Editmenge: WideString
property PreisEinheit: Double
property EinzelPreis: Double
property Rabatt: Double
property Rabatt2: Double
property GesamtPreis: Double
property Gewicht: Double
property KalkpreisEuro: Double
property SteuerProzent: Double
property Provision: Double
property Langtext: WideString
property Bestellnummer:
WideString;
```

8 Werkauftrag-Objekt - IWerkauftrag

Mit diesem Objekt haben Sie Zugriff auf die lagernden- und statusändernden Funktionen der Werkaufträge. Eine Instanz dieses Objektes wird mit der Funktion `GetWerkauftrag()` des Fakt-Objektes erzeugt. Im Folgenden werden Funktionen beschrieben, die analog zu den Schaltern im Werkauftrag entsprechende Aktionen ausführen oder zurücknehmen. Gültige Konstanten für den Übergabeparameter Status sind:

```
sReservieren   = 0
sLagern        = 1
sFertigstellen = 2
```

Funktion	Beschreibung
`Aktion(Status: Integer)`	Führt die Aktion entsprechend des Übergabeparameters aus.
`ResetAktion(Status: Integer)`	Nimmt die Aktion entsprechend des Übergabeparameters zurück.
`SetBeleg(Belegnummer: WideString)`	Lädt Daten eines Werkauftrages entsprechend der übergebenen Belegnummer in das aktuelle Objekt.
`property BelegNummer: WideString`	Zugriff auf die Belegnummer des Werkauftrags. Das Setzen des Properties ruft die Funktion `IWerkauftrag.SetBeleg()` auf.
`SerieChargeListeEinlagern(Datum, Lager, Liste: WideString)`	Lagert eine Liste von Serien- oder Chargennummern für den entsprechenden Werkauftrag ein. Die Angabe des Lagers ist optional, wenn im Werkauftrag ein Lager für Einlagerungen angegeben wurde. Der Parameter Liste wird als String mit Umbrüchen übergeben. Die einzelnen Bestandteile einer Zeile werden durch Semikolon getrennt. Format einer Zeile von „Liste“: `SerienChargenNummer;Menge` ***Achtung:*** *In den Wawi-Versionen 12.0 bis 12.0.2.8 setzte diese Funktion die Fertigungsmengen für "In Produktion" in der Tabelle LAGERFERTIGUNG des Werkauftrages nicht zurück. Der Fehler wurde mit Version 12.0.2.9 korrigiert.*

`function AktionMengeDatum(Status: Integer; Menge: Double; Datum: TDateTime): SYSINT;`	Ausführung von Werkauftragsaktionen mit der Möglichkeit von Mengen- und Datumsangabe.Bei ungültigen Mengenangaben (<= 0 bzw. > Produktionsmenge) wird der Mengendialog bis zur Eingabe einer gültigen Menge angezeigt.Der Parameter Datum wird nur bei der Aktion "fertigstellen" beachtet. Bei den Aktionen "reservieren" und "lagern" wird dieser Parameter ignoriert und wie beim interaktiven Ausführen der Funktionen innerhalb der Wawi das Tagesdatum eingetragen.Der Rückgabewert ist 0 bei fehlerfreier Ausführung der Funktion bzw. -1 bei einem Fehler.
`property Artikelnummer: WideString;`	Rückgabe der Artikelnummer des Werkauftrages.
`property Menge: Double;`	Rückgabe der Menge des Werkauftrages.
`procedure Auslagern(Menge: Double; const AuslagerungsInfo: WideString);`	Auslagern von Stücklistenunterartikeln für die Produktion. Die Funktion führt intern die Funktion `IWerkauftrag.Aktion` mit dem Parameter sLagern (1) aus (Rücknahme von Reservierungen sowie Auslagerung der Unterartikel der Produktionsstückliste - Schalter "Auslagern" im Werkauftrag). Sie bietet darüber hinaus die Möglichkeit, den vom Programm generierten Lagervorschlag zu beinflussen bzw. zu überschreiben. **Parameter:** • *Menge* Die Menge, die bezogen auf die Produktionsstücklistenmenge ausgelagert werden soll. Die Menge muss größer Null und kleiner gleich der momentan im Werkauftrag (offenen) auslagerungsfähigen Menge sein. • *AuslagerungsInfo* Der Parameter AuslagerungsInfo wird als String mit Zeilenumbrüchen übergeben. Die einzelnen Bestandteile einer Zeile werden durch Semikolon getrennt. Format einer Zeile von AuslagerungsInfo: `Kennung; Menge; Lager; SerieCharge; Verfallsdatum` Die Kennung ist der Inhalt des Feldes Kennung der Produktionsstücklistenposition (Datentyp GUID). Die Summe der Mengen je Kennung muß die Menge

ergeben, die bezogen auf den Parameter Menge (siehe oben) und die in der Stücklistenposition des Werkauftrages angegebenen Menge ausgelagert werden soll.

AuslagerungsInfo muß mindestens die Werte für Kennung, Menge und Lager enthalten. Die Bestandteile für SerieCharge und Verfallsdatum sind optional und können einfach weggelassen werden. (Preismengenartikel sind in Werkauftrag-Positionen aktuell nicht zulässig.)

Hinweis:
Es ist nicht notwendig, alle Positionen der Stückliste im Parameter AuslagerungsInfo anzugeben.

Das gilt insbesondere für normale Artikel (keine S/C- oder Verfallsdatumartikel), die mit dem Lagerungsvorschlag des Programms wie vorgesehen ausgelagert werden würden.

Beispiel:
Werkauftrag mit Hauptmenge 1 enthält eine Position mit einem Seriennummernartikel mit Menge 2
Aufruf der Funktion:

```
Werkauftrag.Auslagern(1,
   '{0E25F34E-F7A9-4436-B001-5464770A98FC};1;100;s1'
   + #13#10 +
   '{0E25F34E-F7A9-4436-B001-5464770A98FC};1;100;s2'
);
```

9 Belegtypen-Objekt - IBelegTypen

Funktion	Beschreibung
`property Count: Integer readonly;`	Liefert die Anzahl der Belegtypen zurück.
`property BelegTyp[Index: Integer]: IBelegTyp readonly;`	Liest den Belegtyp an der Position "Index".

10 Belegtyp-Objekt - IBelegTyp

Funktion	Beschreibung
`property BelegTyp: WideString readonly;`	Ermittlung des Belegtypen.

`property Beschreibung: WideString readonly;`	Ermittlung der Beschreibung.
`property UmsatzKennzeichen: WideString readonly;`	Ermittlung des Umsatzkennzeichens.
`property AdressBK: WideString readonly;`	Ermittlung des Adress-Blobkeys.
`property EAFaktor: SYSINT readonly;`	Ermittlung des EA-Faktors.
`property AuswertKennzeichen: WideString readonly;`	Ermittlung des Auswertungskennzeichens.

11 Lager-Objekt - ILager

Funktion	Beschreibung
`function OffeneInventuren: WideString;`	Diese Funktion liefert eine Liste der Belegnummern offener Inventuren zurück (Feld "Belegnummer" der Tabelle "BELEG").
`function InventurIstWert(const InventurNummer: WideString; const Artikelnummer: WideString; const Lager: WideString; const Seriennummer: WideString; Menge: Double): Integer;`	Setzt den Ist-Bestand für einen Artikel in der Inventur. Ab Wawi 9.0.4 werden auch Läger mit Lagerplätzen unterstützt.
`function Lagerbestand(const Artikelnummer: WideString; const Lager: WideString; const Seriennummer: WideString): Double;`	Ermittelt den aktuellen Lagerbestand eines Artikels.
`function GesamtBestand(const Artikelnummer: WideString; const Seriennummer: WideString; MitSperrlager: WordBool): Double;`	Ermittelt den Gesamtbestand je Artikel.

12 Makro-Parameterlisten-Objekt - IMakroParamList

Funktion	Beschreibung
`property Count: Integer;`	Mit dem Property Count kann die Anzahl der Parameter in der Liste abgefragt werden.

`procedure AddParam(const Name: WideString; const Value: WideString);`	Mit dieser Funktion kann ein Parameter zur Makro-Parameterliste hinzugefügt werden. Ein Entfernen von Parametern aus der Liste ist nicht möglich.
	
	Besonderheiten des Datentyps DateTime:
	Bei Parametern vom Datentyp DateTime ist das Datum in doppelte Anführungszeichen zu setzen. Dadurch greifen die internen Mechanismen der SelectLine-Programme zur Konvertierung von Datentypen:
	
	Beispiel:
	
	`makroParamList.AddParam(`
	`  'Param_AngelegtAm', '"01.10.2008"'`
	`);`

`property Name[Index: Integer]: WideString;`
`property Value[Index: Integer]: WideString;`
	Die beiden Properties Name und Value sind die einzige Möglichkeit des Zugriffs auf die Parameternamen und -werte der Makro-Parameterliste. Beide Properties können nur gelesen werden.
	
	Beispiel:
	
	`var`
	`i: Integer;`
	`name: string;`
	`value: string;`
	`...`
	`for i := 0 to makroParamList.Count - 1 do`
	`begin { Voraussetzung: makroParamList enthält bereits Parameter }`
	`  name  := makroParamList.Name[i];`
	`  value := makroParamList.Value[i];`
	`end;`

13 nicht-verwendbare COM-Objekte

Die folgenden COM-Objekte stehen nicht für COM-Anwendungen zur Verfügung. Sie werden für die Anbindung externer Komponenten verwendet (CRM-Outlook-Add-In, Cobra-Add-In). Eine Garantie auf Verfügbarkeit sowie feste Parameterreihenfolge besteht nicht!

- OLEKontaktStatusList
- TOLEAdressList

14 Tipps & Tricks, bekannte Probleme

Dieser Abschnitt enthält Hinweise zu möglichen Problemen und ihren Lösungsmöglichkeiten.

14.1.1 Die Funktion XFakt.ApplicationReady kehrt nie zurück – das COM-Programm scheint zu hängen

Bei einer hohen Belastung des Servers, im konkreten Fall liefen auf dem Server lokale Zugriffe und mehrere Terminalserver-Sessions auf der BDE-Wawi, kehrte das COM-Programm beim Warten auf den Start der Wawi aus der Schleife nie zurück:

```
while not Fakt.ApplicationReady do;
```

Der Grund: Zusätzlich zur hohen Serverbelastung während der Wawi-Initialisierung läuft das COM-Programm in einer Endlosschleife und wartet auf den Wawi-Start, was wiederum die Serverlast unnötigerweise erhöht. Hier empfiehlt es sich, das COM-Programm für eine gewisse Zeit stillzulegen:

```
while not Fakt.ApplicationReady do
    Sleep(100); { Angabe in Millisekunden,
                  hier: eine Zehntelsekunde warten }
```

Der Delphi "Sleep"-Befehl wird direkt an den Microsoft Windows API-Befehl "Sleep" durchgereicht. Dieser bewirkt, dass der Thread seine Zeitscheibe nicht voll ausnutzt, sondern sofort unterbrochen wird, so dass der nächste Thread an die Reihe kommt.

Weitere Informationen finden sich in der MSDN Library in der Beschreibung der Funktionen "`Sleep`" bzw. "`SleepEx`". In anderen Programmiersprachen muß entsprechend nach Funktionen gesucht werden, die diese API-Aufrufe durchführen.

Es ist eine gute Idee, diesen Befehl generell einzubauen. Im Idealfall, wenn die Wawi sofort startet, "verschwendet" man einmalig je nach Parameter ein bißchen Zeit. In den hier in der Hilfe aufgeführten Code-Beispielen wurde der Befehl größtenteils weggelassen, um die Beispiele so einfach wie möglich zu halten.

14.1.2 Über COM aufgerufene Dialoge erhalten keinen Fensterfokus und öffnen sich im Hintergrund

Zur Erklärung dieses "Phänomens" ist etwas "Windows-Geschichte" notwendig:
Bis einschließlich Windows 98 bzw. - auf der ehemaligen "Profi-Schiene" - Windows NT wurden gestartete Programme immer im Vordergrund geöffnet. Das hatte z. B. zur Folge, dass während eines Programmstarts bediente andere Programme (oder das geöffnete Startmenü) plötzlich ihren Fokus verloren und sich das neue Programm nach vorn drängelte.

Ab Windows ME bzw. Windows 2000 hat Microsoft dieses Verhalten schrittweise geändert, so dass in aktuellen Windows-Versionen Programme im Hintergrund gestartet werden. Derart gestartete Programme machen sich seitdem mit einem Blinken in der Taskleiste bemerkbar.

Für die Interoperabilität verteilter Anwendungen mußte jetzt allerdings eine Möglichkeit geschaffen werden, um Dialogfenster gezielt im Vordergrund öffnen zu können:

Der (aufgerufene) Server (->SelectLine Wawi) muß dafür sorgen, dass neue Dialogfenster programmweit im Vordergrund geöffnet werden. Üblicherweise passiert das beim Öffnen neuer Dialoge durch die verwendeten Komponenten oder durch Aufruf der Windows-API-Funktion "`BringToFront`" oder ähnlicher Funktionen.

Der COM-Client wiederum muß dem Betriebssystem mitteilen, dass er anderen Programmen erlaubt, Dialoge im Vordergrund zu öffnen. Dazu muß die Windows-API-Funktion "`AllowSetForegroundWindow`" mit Angabe des Prozeß-Handles oder dem Parameter "aller Prozesse" aufgerufen werden. **Dieser Aufruf ist vor jedem COM-Aufruf sinnvoll, der potentiell Dialoge öffnen könnte.**

In Microsoft Visual Studio befindet sich die Deklaration in WinUser.h.
In Embarcadero Delphi 2009 ist diese Funktion nicht deklariert, dieses muß erst händisch erledigt werden. Der Grund ist die Abwärtskompatibilität von Delphi bis zur Windows Version 95, in der die Funktion noch nicht vorhanden war. In späteren Windows-Versionen eingeführte API-Aufrufe sind deshalb in Delphi nicht deklariert.
In anderen Programmiersprachen und Entwicklungsumgebungen ist entsprechend zu prüfen, ob die Funktion vorhanden ist.
Im SelectLine Wawi-Demo der V. 11.0 (Unit main.pas) ist diese Funktion deklariert und wird an zwei Stellen beispielhaft aufgerufen. Als Parameter dürfen "alle Prozesse" Dialoge im Vordergrund öffnen.

Im "Spezial-COM-Startmodus" INSTANCING=MULTI funktioniert dieses Verhalten nicht zuverlässig, da der SelectLine COM-Server nicht feststellen kann, von welchem COM-Client der aktuelle Aufruf kommt.

Übersicht über die unterschiedlichen Verhaltensweisen:

COM-Aufruf	Windows-Version	Verhalten	Bemerkungen
Funktion `AllowSetForegroundWindow` im Windows-API noch nicht vorhanden	Windows 9x Windows NT	Dialoganzeige immer im Vordergrund	Diese Windows-Versionen werden von SelectLine-Programmen nicht mehr unterstützt.
kein Aufruf von `AllowSetForegroundWindow`	Windows ME alle Versionen ab Windows 2000	blinkendes Symbol in der Taskleiste	Ältere Windows-Versionen werden von SelectLine-Programmen nicht mehr unterstützt.
mit Aufruf von `AllowSetForegroundWindow`	Windows ME alle Versionen ab Windows 2000	Dialoganzeige immer im Vordergrund	Ältere Windows-Versionen werden von SelectLine-Programmen nicht mehr unterstützt.

siehe auch:

- Delphi-Demo ab V. 11.0 (main.pas)
- MSDN-Hilfe zur Funktion AllowSetForegroundWindow

15 Datentypen

15.1 Fehler

Der Datentyp Fehler wird beim Aufruf der Funktion `LetzterFehler()` des Fakt-Objektes zurückgegeben. Er ist als Record aufgebaut und hat folgende Felder:

Nummer: long

Gibt die Nummer des auftretenden Fehlers zurück. Diese Liste kann ständig erweitert werden.

1: Alle vom Programm ausgegebenen Meldungen und Hinweise
2: Kreditlimit überschritten
weitere können folgen

Klasse: Fehlerklasse

Meldung: Alle Programmhinweise, die mit ‚OK' bestätigt werden können.
Frage: Alle vom Programm ausgegebenen Fragen
Abbruch: Alle Meldungen vom Programm, bei denen anschließend der normale Programmablauf unterbrochen wird.
KeinFehler: Beim Ausführen der letzten Funktion ist kein Fehler aufgetreten.

Meldung: WideString

Enthält den Meldungstext des Dialogfensters.

15.2 Fehlerklasse

Datentyp für Feld Klasse im Typ Fehler (Meldung, Frage, Abbruch, KeinFehler)

15.3 Erläuterung des Datentyps Variant:

Einige Funktionen können mit variantem Datentyp aufgerufen werden oder liefern einen solchen zurück. Varianten können ganze oder reelle Zahlen, Stringwerte, Boolesche Werte, Datum-Uhrzeit-Werte sowie COM-Automatisierungsobjekte enthalten.

16 Anhang

16.1 Steuerung der Wawi-Instanziierung

Ab der Warenwirtschaft Version 8.1.3 ist es möglich, das Instanziierungsverhalten bei COM-Aufrufen über einen Eintrag in der FAKT.INI zu steuern. Standardverhalten ist, dass bei jeder Initialisierung eines XFakt.App-COM-Objektes eine neue Instanz der Warenwirtschaft gestartet wird. Mit dem folgenden INI-Eintrag wird erreicht, dass alle COM-Programme von einer Instanz bedient werden.

In der Datei FAKT.INI:

```
[COM]
INSTANCING=MULTI
```

Standard ist: fehlender Eintrag bzw. `INSTANCING=SINGLE`

Mit diesem INI-Eintrag ergeben sich folgende Kombinationsmöglichkeiten bei "normalem" Wawi-Start und Start über COM:

erster Start der Wawi	Eintrag in FAKT.INI	weitere Starts der Wawi(s)	Erläuterungen
direkter Aufruf	kein Eintrag	Jeder Wawi-Aufruf (direkt oder über COM) startet eine neue Instanz. Anzahl Instanzen: Anzahl Direktstarts + Anzahl COM-Starts	Standardverhalten ohne INI-Eintrag
direkter Aufruf	MULTI	Alle COM-Anforderungen werden von der bereits laufenden Wawi-Instanz bedient. Anzahl Instanzen: Anzahl Direktstarts	
Start über COM	kein Eintrag	Jede COM-Anforderung startet eine neue Wawi-Instanz. Anzahl Instanzen: Anzahl Direktstarts + Anzahl COM-Starts	Standardverhalten ohne INI-Eintrag
Start über COM	MULTI	Alle COM-Anforderungen werden von der ersten gestarteten COM-Instanz bedient. Anzahl Instanzen: 1 COM-Instanz + Anzahl Direktstarts	

16.2 Liste der wichtigsten Tabellenschlüssel

Schlüssel	Tabellenname	Beschreibung
AR	ART	Stammdaten Artikel
BA	BANKEN	Stammdaten Banken
KB	KASSBANK	Stammdaten Bankbezüge

KU	KUNDEN	Stammdaten Kunden
LI	LIEFER	Stammdaten Lieferanten
QA	BELEG	Angebote Kunden
QB	BELEG	Bestellungen
QF	BELEG	Aufträge
QR	BELEG	Rechnungen
WA	WAEHR	Stammdaten Währungen
WP	WAEHRP	Stammdaten Währungen (Detailtabelle)
ZB	ZAHLBED	Stammdaten Zahlungsbedingungen

Eine vollständige Liste kann in der Warenwirtschaft (Programmpunkt Extras - Vorgaben) eingesehen werden.